Jonathan Skaggs

CS 478 Decision Tree

When would you get 100% training accuracy

You will often get 100% on the training data if you do not prune because it over-fits to the data. The only case where this will not be the case is if there is no pruning and there as least one case that two instances have all the same features but a different label.

	Cars		Voting	
	Train Acc	Test Acc	Train Acc	Test Acc
Fold 0	100.00%	64.53%	96.17%	90.70%
Fold 1	96.08%	100.00%	95.40%	97.73%
Fold 2	96.08%	100.00%	95.66%	95.35%
Fold 3	96.08%	100.00%	95.40%	97.73%
Fold 4	96.08%	100.00%	95.41%	97.67%
Fold 5	96.08%	100.00%	95.14%	100.00%
Fold 6	96.08%	100.00%	96.17%	90.70%
Fold 7	96.08%	100.00%	95.40%	97.73%
Fold 8	96.08%	100.00%	95.92%	93.02%
Fold 9	96.08%	100.00%	95.65%	95.45%
Mean	96.47%	96.45%	95.63%	95.61%

What I observed

I noticed that when I used the entire training set without any pruning, both accuracies (training and testing) were very high. When the entire tree is populated there is enough data to predict the testing set more accurately. This is because the more instances there are, in the training set, at each node helps to suppress outliers.

What the cars dataset has learned

The number one telling attribute for the cars dataset was the number of doors. (The tree learned to split on "doors" first). Following this the next thing to split on was the size of the "lug_boot". I did think it was interesting that the cars dataset learned that if there was only one door then instead of splitting on the size of the "lug_boot" it would first split on "maint". After the first three layers of the tree it is hard to write what comes next because it varies so much from node to node, however, in most cases "buying" or "persons" were chosen next followed by "safety". As it turns out for this particular data set "safety" was the least telling of all of the features. In other words, it helped the least when trying to predict the label.

What the voting dataset has learned

In the voting dataset, the decision tree learned that how people voted on "mx-missile" was the most telling of their political party. Secondly, both paths split on "handicapped-infants". After this point, it becomes difficult to follow because the next nodes all split on different issues. The next issues chosen were "immigration", "religious-groups-in-schools", "el-salvador-aid", and "class_name". After this, it becomes hard to tell what is most important because the tree is so large and each node splits on so many different combinations of the order of features.

What I did with unknown data and Why I did

I decided that I did not think in general it make sense to make unknown data its own separate branch. Although it could be an indication of the party in this situation, because unknown could mean did not vote, I think that in general this is not going to be the case. To me it made more sense to push what we did know about the data down the tree so that even if there was missing data in an instance, that instance would still contribute to the knowledge and accuracy of the sub-nodes that it did know about. When it came time to testing, if I came to an unknown value, I would pass the instance down both sides of the tree and calculate the probability that the label is democrat or republican. I would then choose the highest label. I did this calculation by calculation the probability that the label for the instance was "democrat" given that the missing value was "y" plus the probability that the label for the instance was "democrat" given that the missing value was "n". I would then do the same thing for republican. Then I would choose the probability that was more likely.

Pruning the Data

	Cars		Voting	
	Original	Pruning	Original	Pruning
# of Nodes	2524	2209	2325	1321
Test Acc	96.47%	94.19%	95.63%	95.41%

Going Farther

When doing reduced error pruning I originally pruned by looping through and testing what happens when we temporarily remove each node. If temporarily removing the node did not decrease my accuracy, then I would permanently delete it. I decided that this process is highly inefficient so I decided that for my experiments I would try a different way. My though is that rather than check the change in accuracy of the validation set for entire tree just check the change in accuracy for each sub node. (It is important to note that my code is vectorized, so I pass all relevant instances down the tree). Using a DFS, starting with one branch above the leaf nodes I sum up how many number of instances that the children guess correctly and divide by the number of total number of instances. If the accuracy of the children is not better than the adult node then they are clipped. You continue to do this traversing the tree in a depth first search. When I ran the code this way it got the same accuracies on the test set. It was much faster than testing the whole tree 2200 times.